Diameters of Permutation Groups on Graphs and Linear Time Feasibility Test of Pebble Motion Problems

Jingjin Yu

***Abstract*— Let G be an n-vertex connected, undirected, simple graph. The vertices of G are populated with n uniquely labeled pebbles, one on each vertex. Allowing pebbles on cycles of G to rotate (synchronous rotations along multiple disjoint cycles are permitted), the resulting pebble permutations form a group G uniquely determined by G. Let the diameter of G (denoted $diam(\mathbf{G})$) represent the length of the longest product of generators (cyclic pebble rotations) required to reach an element of G, we show that $diam(\mathbf{G}) = O(n^2)$.**

Extending the formulation to allow $p \leq n$ pebbles on an n-vertex graph, we obtain a variation of the (classic) pebble motion problem (first fully described in Kornhauser, Miller, and Spirakis [7]) that also allows rotations of pebbles along a fully occupied cycle. For our formulation as well as the (classic) pebble motion problem, given any start and goal pebble configurations, we provide a linear time algorithm that decides whether the goal configuration is reachable from the start configuration. This gives a positive answer to an open problem raised by Auletta et al. in [1].

I. Introduction

In Sam Loyd's 15-puzzle [8], a player is asked to arrange square game pieces labeled 1-15, scrambled on a 4×4 grid, to a shuffled row major ordering, using one empty swap cell: In each step, one of the labeled pieces neighboring the empty cell may be moved to the empty cell (see, e.g., Fig. 1). Generalizing the grid to an arbitrary connected graph over n vertices, the 15-puzzle becomes the pebble motion problem on graphs in which one is asked to sequentially move as many as $n-1$ uniquely labeled pebbles on the vertices of the graph to some desired goal configuration, using unoccupied (empty) vertices as swap spaces. In this paper, we focus on a generalized version of the pebble motion problem: Besides allowing uniquely labeled pebbles to move using empty vertices of a graph, rotations of pebbles on any cycle fully occupied by pebbles are also permitted. Our formulation provides a better model for problems such as multi-robot motion planning and data routing on arbitrary graphs, for the following reasons: 1. In general, simultaneous movements of entities (i.e., robots or data) are desirable; 2. In particular, cyclic movements of these entities on fully occupied cycles should be allowed.

As early as 1879, Story [11] observed that the parity of a 15-puzzle instance decides whether it is solvable. Wilson [15] formalized this observation by showing that the reachable configurations of a 15-puzzle form an alternating

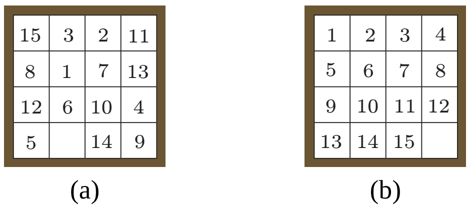

Fig. 1. Two 15-puzzle instances. a) An unsolved instance. In the next step, one of the pieces labeled 5, 6, 14 may move to the vacant cell, leaving behind it another vacant cell for the next move. b) The solved instance.

group on 16 letters. He also provided an algorithm for producing a solution for a solvable instance. Kornhauser, Miller, and Spirakis [7] replaced the potentially exponential length algorithm in [15] with a polynomial time algorithm, which produces solutions with a $O(n^3)$ upper bound on the number of moves, for general graphs with n vertices and up to $n-1$ pebbles. Auletta et al. [1] later showed that for trees, deciding whether an given instance of the pebble motion problem is feasible can be done in linear time.

From the perspective of optimality, Goldreich [6] proved that computing a shortest solution (in terms of the number of moves) for the the pebble motion problem is NP-hard. Ratner and Warmuth [10] showed that finding a shortest solution for the generalization of the 15-puzzle, called (n^2-1)-puzzle, is also NP-hard (note that this problem is a restriction of the pebble motion problem). More recently, Surynek [12] extends the NP-hardness result to pebble motion problems in which multiple pebble moves per step are allowed.

As evident from the techniques used in [7], [15], the pebble motion and related problems are closely related to structures of permutation groups. Fixing a graph and the number of pebbles, and viewing the pebble moving operations as generators, all configurations reachable from an initial configuration forms a group that is isomorphic to a subgroup of $\mathbf{S_n}$, the symmetric group on n letters. Deciding whether a problem instance is feasible is then equivalent to deciding whether the final configuration is reachable from the initial configuration via generator products. Another interesting problem in this domain is the study of the *diameter* of such groups, which is the length of the longest generator product required to reach a group element. Driscoll and Furst [4], [5] showed that any group represented by generators that are cycles of bounded degree has a diameter of $O(n^2)$ and such a generator sequence is efficiently computable. For generators of unbounded size, Babai, Beals, and Seress [2] proved that if one of the generators fixes at least 67% of the

Jingjin Yu is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. E-mail: jyu18@uiuc.edu. This research was supported in part by NSF grant 0904501 (IIS Robotics), NSF grant 1035345 (Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

domain, then the resulting group has a polynomial diameter. In contrast, groups with super polynomial diameters exist [4].

We begin our study (in Section III) by looking at the groups generated by cyclic rotations of labeled pebbles on graphs fully occupied by pebbles. We show that such groups have $O(n^2)$ diameters. This result is not implied by [2], [4] since there are cases in which all generators are of unbounded size and fix less than 67% of the domain. With this intermediate result, we continue to show, in Section IV, that the feasibility test of the *pebble motion with rotation* problem can be performed in $O(|V|+|E|)$ time, which implies a $O(n^3)$ algorithm for computing a feasible solution (the set of movements). We then note that our feasibility test can be modified to yield a linear time feasibility test for pebble motion on graphs, which provides a positive answer to an open question raised in [1] asking whether a linear time feasibility test for pebble motion on trees can be extended to general graphs. Finally, in Section V, we discuss NP-hardness of finding optimal solutions and lower bounds on group diameters generated by cyclic rotations.

II. Preliminaries

A. Pebble motion problems.

Let $G=(V,E)$ be a connected, undirected, simple graph[1] with $|V|=n$. Let there be a set $p \leq n$ pebbles, numbered $1,\ldots,p$, residing on distinct vertices of G. A *configuration* of these pebbles is a sequence $S=\langle s_1,\ldots,s_p\rangle$, in which s_i denotes the vertex occupied by pebble i. Such a configuration can also be viewed as a bijective map $S:\{1,\ldots,p\}\rightarrow V(S)$ in which $V(S)$ denotes the set of vertices in S. We allow two types of *moves* of pebbles: A *simple move*, in which a pebble moves to an empty vertex or a *rotation*, in which pebbles occupying all vertices of a cycle rotate simultaneously (clockwise or counterclockwise) so that each pebble moves to the vertex previously occupied by its (clockwise or counterclockwise) neighbor. Two configurations S,S' are *connected* if there exists a sequence of moves that takes S to S'. Let S,D be two pebble configurations on a given graph G, the *pebble motion* problem or PMG is defined as follows.

Problem 1 (Pebble Motion on Graphs) *Let (G,S,D) be an instance of PMG. Find a sequence of simple moves that take S to D.*

When G is a tree, PMG problems are also referred to as *pebble motion on trees* or PMT. In this case, an instance is usually written as $I=(T,S,D)$. When both simple moves and rotations are allowed (one move per time step), the resulting variant is the *pebble motion with rotation* problem or PMR.

Problem 2 (Pebble Motion with Rotation) *Let (G,S,D) be an instance of PMR. Find a sequence of simple moves and rotations that takes S to D.*

[1]The presented readily generalize to graphs with multiple connected components (via taking the direct product of the groups from the components). Generalizations to directed graphs and graphs with multiple edges are also straightforward.

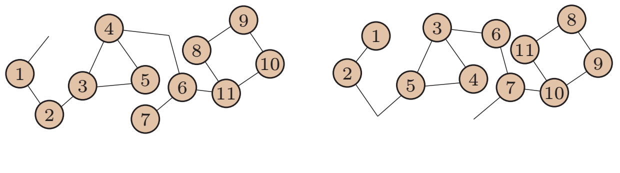

Fig. 2. Two pebble configurations that are connected via a single *synchronous move*.

Note that if G is a tree an PMR instance is simple a PMT instance. We mentioned in the introduction that PMR is a better model for problems such as multi-robot motion planning and data routing problems. We now make this claim more precise. A *synchronous move* is a set of moves that can be carried out in the same time step such that no two pebbles will travel along the same edge of G. This allows concurrent simple moves as long as each pebble is moving to a vertex that is vacated in the same step, as well as concurrent rotations of pebbles on multiple disjoint cycles. An illustration of a synchronous move is given in Fig. 2. We define the *synchronous pebble motion* problem or SPM as follows.

Problem 3 (Synchronous Pebble Motion) *Let (G,S,D) be an instance of SPM. Find a sequence of synchronous moves that takes S to D.*

Since in applications such as multi-robot motion planning, individual entities can communicate and therefore are capable of synchronous motions including rotations along a fully occupied cycle, SPM provides a more realistic model for these practical problems. On the other hand, it is clear that an instance $I=(G,S,D)$ of PMR is feasible if and only if I, interpreted as an instance of SPM is also feasible: Given a sequence of moves that solves a PMR problem, it is also a solution to the corresponding SPM problem. It is also straightforward to "compress" the solution to get a more compact solution to the SPM problem. On the other hand, a solution to an SPM instance can be "serialized" to obtain a solution to the corresponding PMR problem.

A natural criterion for measuring solution optimality of SPM problem is the number of time steps (i.e., synchronous moves) it takes until the goal configuration is reached, which is often referred to as the *makespan*. Let $X=\langle S=S_0,S_1,\ldots,D\rangle$ be a sequence of configurations in which each pair of consecutive configurations S_t,S_{t+1} are connected via a synchronous move, X is then a solution to SPM. Let $\mathscr{X}$ denote all solutions to an SPM instance. We may state the following decision problem:

TOSPM (Time Optimal Synchronous Pebble Motion)
INSTANCE: An instance of SPM and $k\in\mathbb{Z}$.
QUESTION: Is there an $X\in\mathscr{X}$ such that $|X|\leq k$?

B. Groups generated by cyclic pebble motions.

A particularly important case of SPM is when $p=n$; we restrict our discussion to this case in this subsection and

Sections III. When $p = n$, the only possible motions are synchronous rotations. Given two configurations S, S' that are connected, they induce a permutation of the pebbles, which is computable via $\sigma_{S,S'}(i) = S^{-1}(S'(i))$ for each pebble i; $\sigma_{S,S}$ yields the identity permutation. Given an initial configuration S_0, let $\mathscr{S}$ denote the set of all configurations reachable from S_0. It can be verified (using the basic definition of a group) that the permutations σ_{S_0,S_i} over all $S_i \in \mathscr{S}$ form a subgroup of $\mathbf{S_n}$, the symmetric group on n letters. Since it is clear the this group is determined by the graph G, we denote it $\mathbf{G}$.

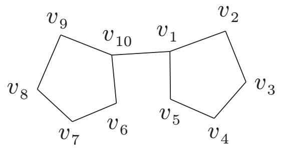

Fig. 3. For the graph above, the set of sets of cycles are $\mathscr{C} = \{\{v_1v_2v_3v_4v_5\}, \{v_6v_7v_8v_9v_{10}\}, \{v_1v_2v_3v_4v_5, v_6v_7v_8v_9v_{10}\}\}$.

Two cycles, as subgraphs of G, are *disjoint* if their vertex sets do not overlap. When $p = n$, each synchronous move corresponds to the rotations of pebbles along a set of of disjoint cycles. Let $\mathscr{C}$ be the set of all sets of disjoint cycles in G; each $C \in \mathscr{C}$ is a unique set of disjoint cycles of G. Since the pebbles may rotate clockwise or counterclockwise along a cycle $c_i \in C$, each set of disjoint cycles C can take a configuration to $2^{|C|}$ new configurations with one move. That is, each C yields $2^{|C|}$ generators of $\mathbf{G}$. Let the set of all generators obtained this way be $\mathscr{G}$, a finite set. As an example, the graph in Fig. 3 has two cycles, with $|\mathscr{C}| = 3$ and $|\mathscr{G}| = 8$. We make the simple observation that these definitions yield a natural bijection between synchronous moves and elements of $\mathscr{G}$. As such, when a configuration S' is reachable from a configuration S, we say that the permutation $\sigma_{S,S'} \in \mathbf{G}$ is *reachable* (from the identity) using products of generators from $\mathscr{G}$ corresponding to the synchronous moves. We frequently invoke this bijection between synchronous moves and generators without explicitly stating so. The following is obvious (the simple fact that $1 + |\mathscr{G}| + \ldots + |\mathscr{G}|^k \leq |\mathscr{G}|^{k+1}$ is used; note that $|\mathscr{G}| \neq 1$).

Lemma 4 *When $p = n$, starting from a configuration S_0, in k synchronous moves, at most $|\mathscr{G}|^{k+1}$ different configurations are reachable. Alternatively, using a generator product of length k, at most $|\mathscr{G}|^{k+1}$ elements of $\mathbf{G}$ are reachable.*

The smallest k after which no new configuration can be obtained is the *diameter* of $\mathbf{G}$, denoted $diam(\mathbf{G})$.

III. Upper Bounds on Group Diameters

In this section we establish $diam(\mathbf{G}) = O(n^2)$. We divide possible G's in to classes based on connectivity. When G is connected (1-connected) but none of its subgraphs are 2-connected (i.e., G has no cycles), it is a tree. In this case, no pebble can move.

Lemma 5 (Tree) *If G is a tree, then $\mathbf{G} \cong \{1\}$, the trivial group.*

Another simple case is when G is a cycle, the simplest 2-connected graph: It is clear that all of $\mathbf{G}$'s elements are generated by a single rotation (and its inverse). In what follows, $\mathbb{Z}/n$ denotes the cyclic group of order n.

Lemma 6 (Cycle) *If G is a cycle, then $\mathbf{G} \cong \mathbb{Z}/n$.*

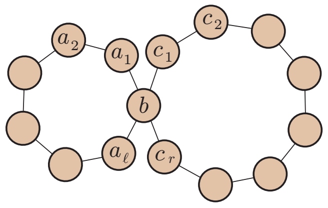

Fig. 4. Two cycles sharing one common vertex. The graph is separable at the vertex occupied by pebble b.

When G is connected but the removal of certain vertices (called *articulation* vertices) from G leaves two or more components, it is *separable*. An important case here is when G is a set of cycles sharing vertices such that no edge of G is on more than one cycle (a *cactus graph* with no *spines*). Such graphs form a subset of 2-edge-connected graphs. Fig. 4 gives an example with two cycles. Following convention, $\mathbf{A_n}$ denotes the *alternating group* on n letters.

Proposition 7 (Cycles, Separable) *If every edge of a separable graph G is on exactly one cycle, then $\mathbf{G} \geq \mathbf{A_n}$ and $diam(\mathbf{G}) = O(n^2)$.*

PROOF. Given configurations S, D (we do not require that D be reachable from S), we claim:

1. In $O(n^2)$ moves, D can be taken to some configuration D' such that D' has all the pebbles on a cycle they belong to in S. If this feels convoluted, an example should clear the confusion. In Fig. 4, assuming the given configuration is S, this step ensures that pebbles a_i's are all on the left cycle and pebbles c_i's on the right cycle. The pebble b may appear on either one of the two cycles.

2. Again in $O(n^2)$ moves from D' yielding another configuration D'', we have either $D'' = S$ or they differ by a transposition (we require that this transposition is fixed for a fixed S).

These claims are proved in lemmas that follow. Following the second claim, since D'' differs from S by at most a fixed transposition, an arbitrary D can reach either S or S', which is the configuration obtained by letting the fixed transposition act on S. Looking at this in reverse, an arbitrary D is then reachable from either S or S'. Therefore, all configurations (and consequently elements of $\mathbf{S_n}$) are partitioned into two equivalence classes based on mutual reachability. Since the only subgroup of $\mathbf{S_n}$ of index 2 is $\mathbf{A_n}$, this implies that $\mathbf{G} \geq \mathbf{A_n}$.

When $\mathbf{G} \cong \mathbf{A_n}$, any element of $\mathbf{G}$ is a product of generators from $\mathscr{G}$ with a length of $O(n^2)$, proving $diam(\mathbf{G}) = O(n^2)$. If $\mathbf{G}$ is not isomorphic to $\mathbf{A_n}$, since the only subgroups of $\mathbf{S_n}$ containing $\mathbf{A_n}$ are $\mathbf{A_n}$ and $\mathbf{S_n}$ itself, $\mathbf{G} \cong \mathbf{S_n}$. This implies that $\mathbf{A_n}$ has at most two cosets in $\mathbf{G}$; denote the other coset of $\mathbf{A_n}$ as ${\mathbf{A_n}}^c$, which also have a diameter of $O(n^2)$ (to see this, note that any configuration D is reachable from one of S, S' in $O(n^2)$ moves). From the identity, all elements of $\mathbf{A_n}$ are reachable using generator products of length $O(n^2)$. Since elements of ${\mathbf{A_n}}^c$ are now reachable from elements of $\mathbf{A_n}$, an element of ${\mathbf{A_n}}^c$ must be reachable from the identity using a generator product of length $O(n^2)$ as well. Therefore, when $\mathbf{G} \cong \mathbf{S_n}$, all elements of $\mathbf{G}$ are reachable using generator products of length $O(n^2)$, yielding $diam(\mathbf{G}) = O(n^2)$. □

Before moving to the lemmas, we note that when G is separable and every edge of G is on exactly one cycle, the edges of G can be partitioned into equivalence classes based on the cycles they belong to. Because G is separable, every cycle must border one or more cycles and at the same time, two cycles can share at most one vertex. Moreover, there exists a cycle that only shares one vertex with other cycles (otherwise, the cycles must form one or more larger cycles, contradicting the assumption that each edge is on exactly one cycle). Call such a cycle a *leaf cycle*. An example of a leaf cycle is given in Fig. 5.

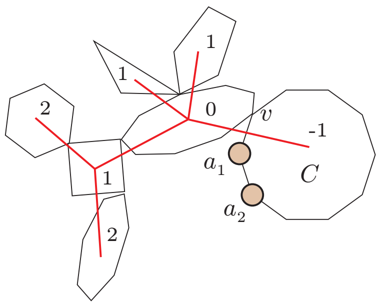

Fig. 5. The dual tree structure in a separable graph G with every edge on exactly one cycle. The numbers represent the cycle distances of the cycles to the leaf cycle C, which in fact is the root of the tree.

Given a cycle C' on G, it is of *cycle distance* d_c to C if a vertex on C' needs to travel through at least d_c cycles to reach C (a neighboring cycle of C has distance 0 since they share a common vertex). Let C have a cycle distance of -1 by definition. This induces a (dual) tree structure on the cycles when viewing them as vertices joined by edges to neighbors. See Fig. 5 for an example of this tree structure. Computing such a tree takes time $O(|V| + |E|)$ (obtaining maximal 2-connected components takes linear time [13]). The first claim in the proof of Proposition 7 can be stated as follows.

Lemma 8 (Initial Arrangement) *Given a separable G with each edge on exactly one cycle and configurations S, D, in $O(n^2)$ moves, all pebbles can be moved (from D) to some cycles they belong to in S.*

PROOF. Note that a pebble may reside on multiple cycles; this lemma only ensures that such a pebble gets moved to one of the cycles it belongs to in S. First we show that a single pebble can be relocated to a cycle it belongs to in S in $O(n)$ rotations, without affecting pebbles that are previously arranged. When G is two cycles joined on a common vertex (e.g., Fig. 4), without loss of generality, assume that we need to move a_i from the left cycle to the right cycle. This implies that some pebble c_j (and possibly b) does not belong to the right cycle in S. For moves, we note that the group $\mathbf{G}$ in this case has four generators,

$$g_\ell = \begin{pmatrix} a_1 & a_2 & \ldots & a_\ell & b \\ b & a_1 & \ldots & a_{\ell-1} & a_\ell \end{pmatrix},$$
$$g_r = \begin{pmatrix} c_1 & c_2 & \ldots & c_r & b \\ c_2 & c_3 & \ldots & b & c_1 \end{pmatrix},$$

which correspond to clockwise rotations along the left and right cycles, respectively, and their inverses, g_ℓ^{-1} and g_r^{-1}. One can verify that the generator product $g_\ell^{-i} g_r^{-j} g_\ell^{i}$ exchanges a_i and c_j between the two cycles without affecting cycle membership of other pebbles (see Fig. 6).

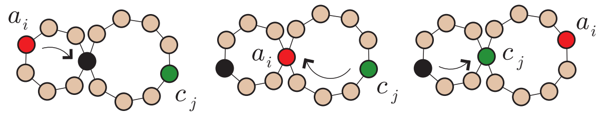

Fig. 6. Illustration of the vertex arrange algorithm for two adjacent cycles.

For the general case in which a pebble needs to go along some k cycles, denoting the generators as $g_1, \ldots, g_k$, it is easy to verify that a product of the form $g_1^{-i_1} g_2^{-i_2} \ldots g_k^{i_k} \ldots g_2^{i_2} g_1^{i_1}$ achieves what we need, with $i_1 + \ldots + i_k < n$ (there may be more than these $2k$ basic generators, but we do not need the other generators for this proof). Therefore, at most $2n$ moves are needed to move one pebble to the desired cycle. To avoid affecting pebbles that are previously arranged, we may simply fix a leaf cycle C and start with cycles based on their cycle distance to C in decreasing order. At most $2n^2$ moves are required to arrange all n pebbles to the desired cycles. □

Lemma 9 (Rearrangement) *The pebbles arranged according to Lemma 8 can be rearranged such that the resulting configuration is the same as S or differ from S by a fixed transposition of two neighboring pebbles in S. Rearrangement requires $O(n^2)$ moves.*

PROOF. For a fixed G, let C be a leaf cycle and let C border other cycle(s) via vertex v. In S, let a_1 be the pebble occupying counterclockwise neighboring vertex of v on the cycle C, and let a_2 be the counterclockwise neighbor of a_1 on C (again, see Fig. 5 for an illustration of this setup). The fixed transposition will be $(a_1 a_2)$.

We rearrange pebbles to match the configuration S starting from cycles with higher cycle distances to the leaf cycle C, using the neighboring cycle with smaller cycle distance (such a cycle is unique). We show that the pebbles on the more distant cycle can always be rearranged to occupy the vertex specified by S. Moreover, this can be achieved using moves that only affect the ordering of two pebbles on the neighboring cycle. Without loss of generality, we use the two cycle example from Fig. 4 and let the right cycle be the more distant one. The generators g_ℓ, g_ℓ^{-1}, g_r, and g_r^{-1} from previous lemma remain the same. To exchange two pebbles on the right cycle, for example c_i, c_j, we may use the following generator product

$$g_\ell^{-2} g_r^{-i} g_\ell g_r^{j-i} g_\ell^{-1} g_r^{-j+i} g_\ell g_r^{-i} g_\ell. \qquad (1)$$

It is straightforward to verify that (1) works. To make it clear, Fig. 7 illustrates the application of (1) for exchanging c_2 and c_5 using a_1, a_2. Every such exchange requires at most $2n$ moves.

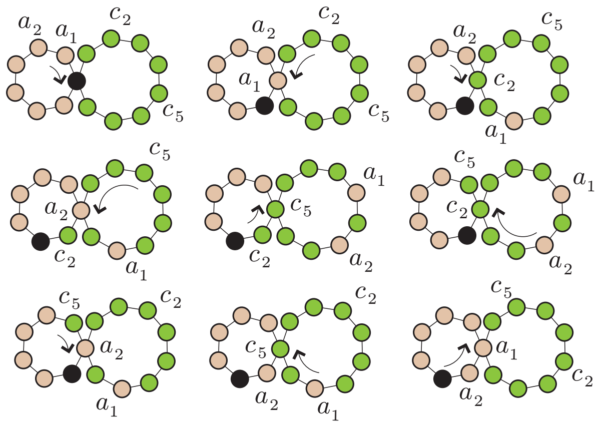

Fig. 7. Illustration of the rearrangement algorithm (from left to right, then top to bottom).

Performing such exchanges iteratively, within $2n^2$ moves, all pebbles except those on the leaf cycle C can be rearranged to occupy vertices specified by S. Reversing the process, we can arrange all pebbles on C to occupy vertices specified by S, using a neighboring cycle C', affecting the ordering of at most two pebbles on C'. Repeating this process again with C' using C as the neighboring cycle and a_1, a_2 as the swapping pebbles, all pebbles except possibly a_1, a_2 occupy the vertices specified by S. □

The above two lemmas complete the proof of Proposition 7. At this point, it is easy to see that when G is separable with each edge on a single cycle, $\mathbf{G} \cong \mathbf{S_n}$ if and only if G contains an even cycle, corresponding to the composition of an odd number of transpositions. Otherwise, $\mathbf{G} \cong \mathbf{A_n}$. We are left with the case in which G is 2-connected but not a (single) cycle.

Proposition 10 (2-connected, General) *If G is 2-connected and not a cycle, $\mathbf{G} \cong \mathbf{S_n}$ with $diam(\mathbf{G}) = O(n^2)$.*

PROOF. Our proof again starts by showing that the locations of two pebbles can be exchanged without affecting the locations of other pebbles. Given a 2-connected graph G that is not a cycle, it can always be decomposed into a cycle plus one or more *ears* (an ear is a simple path P whose two end points lie on some cycle that does not contain other vertices of P). Therefore, any two pebbles on G must lie on some common cycle with one attached ear. We may then assume that the two pebbles to be exchanged lie somewhere on two adjacent cycles (i.e., they are two arbitrary pebbles in Fig. 8). Restricting to such a graph G' of G, which has three cycles (left, right, and outer), rotations along these cycles will not affect the rest of the pebbles not on G'. We claim that moving within G' is sufficient to exchange any two pebbles on G' and the operation can be done with $O(n)$ moves.

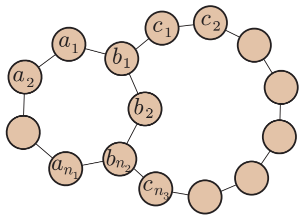

Fig. 8. A simple 2-connected graph. There are six moves for this configuration: Rotating clockwise or counterclockwise along one of the three cycles.

Let G' have $n_1 + n_2 + n_3$ vertices, with n_1 vertices belonging to the left cycle only, n_3 vertices belonging to the right cycle only and n_2 vertices shared by the two cycles. Assuming the initial pebble configuration is as illustrated in Fig. 8, we have the following generators,

$$g_\ell = \begin{pmatrix} a_1 & a_2 & \ldots & a_{n_1} & b_{n_2} & \ldots & b_1 \\ b_1 & a_1 & \ldots & a_{n_1-1} & a_{n_1} & \ldots & b_2 \end{pmatrix},$$
$$g_r = \begin{pmatrix} c_1 & c_2 & \ldots & c_{n_3} & b_{n_2} & \ldots & b_1 \\ c_2 & c_3 & \ldots & b_{n_2} & b_{n_2-1} & \ldots & c_1 \end{pmatrix},$$
$$g_o = \begin{pmatrix} b_1 & c_1 & \ldots & c_{n_3} & b_{n_2} & a_{n_1} & \ldots & a_1 \\ c_1 & c_2 & \ldots & b_{n_2} & a_{n_1} & a_{n_1-1} & \ldots & b_1 \end{pmatrix},$$

which are clockwise rotations along the left, right, and the outer cycles of G', and their inverses, g_ℓ^{-1}, g_r^{-1}, and g_o^{-1}. Note that

$$g_r g_\ell g_o^{-1} = \begin{pmatrix} b_1 & c_1 \\ c_1 & b_1 \end{pmatrix} = (b_1 c_1). \qquad (2)$$

That is, we may exchange (transpose) b_1 and c_1 using a generator product of length 3. Using this length 3 product $g_r g_\ell g_o^{-1}$, it is possible to exchange any two pebbles on G' without affecting other pebbles. We elaborate two such cases, all other cases are similar. In a first case we exchange a_i and c_j. To do this, we first move c_j to c_1's location, followed by moving a_i to b_1's location. We can then switch a_i and c_j using the primitive $g_r g_\ell g_o^{-1}$. Reversing the earlier steps then switches a_i and c_j without affecting any other pebbles. The complete product sequence is $g_\ell^{-i} g_r^j g_\ell g_o^{-1} g_r^{-j+1} g_\ell^i$, which requires $O(n)$ moves or generator actions. Similarly, if we want to switch some c_i, c_j that are not adjacent, we can move them along the outer cycle until one of them belongs to the left cycle and the other to the right cycle. The case of exchanging a_i, c_j then applies, after which we reverse the earlier moves on the outer cycle to obtain the net effect of switching c_i, c_j. The number of moves is again $O(n)$. This implies $\mathbf{G} \cong \mathbf{S_n}$ and $diam(\mathbf{G}) = O(n^2)$. □

Combining Proposition 7 and 10 concludes the case for 2-edge-connected graphs that are not single cycles; the case of general graph then follows. Since we will mention "2-edge-connected component" fairly often, we abbreviate it to "TECC" except in the statement of a result. Also, we call each component of G after deleting all TECCs a *branch*.

Proposition 11 (2-edge-connected) *If G is 2-edge-connected and not a single cycle, $\mathbf{G} \geq \mathbf{A_n}$ with $diam(\mathbf{G}) = O(n^2)$.*

PROOF SKETCH. A 2-edge-connected graph G can be separated into 2-connected components via splitting at articulating vertices. A (dual) tree structure, similar to that illustrated in Fig. 5, can be built over these components. The two-step algorithm used in the proof of Proposition 7, in combination with Proposition 10, can be applied to show that $\mathbf{G} \geq \mathbf{A_n}$ and $diam(\mathbf{G}) = O(n^2)$. □

Theorem 12 (General Graph) *Given an arbitrary connected, undirected, simple graph G, $diam(\mathbf{G}) = O(n^2)$.*

PROOF. A vertex of G that is not on any cycle is always fixed; deleting these vertices does not change $\mathbf{G}$. After all such vertices are removed, we are left with the TECCs of G. Denoting the associated groups of these components $\{\mathbf{G}_i\}$, $\mathbf{G}$ is the direct product of the $\mathbf{G}_i$'s. Since all $\mathbf{G}_i$'s have $O(n^2)$ diameter, so does $\mathbf{G}$. □

As a last note, if we let the graph in Fig. 8 have $n = 2k+3$ vertices for some integer k and let the left and right cycles both have $k+3$ vertices, then all six generators move more than $n/2$ of the domain. Therefore, results from [2], [4] do not lead to a polynomial sized diameter. On the other hand, using (1) with $i = 1, j = 2$ and (2), we can obtain composite generators that fixes more than 67% of the domain. A polynomial size diameter is then implied by [2].

IV. LINEAR TIME FEASIBILITY TEST OF PMR

We now describe an linear time algorithm for testing the feasibility for PMR, using a proof strategy similar to that from [1] on the PMT problem. We start by restating Theorem 3 from [1], which reduces an instance (T, S, D) of PMT to another one (T, S', D), in which the set of vertices in S' and D are the same.

Theorem 13 *Given an instance (T, S, D) of PMT, in $O(n)$ time, an instance (T, S', D) of PMT can be computed such that S', D are the same set of vertices. Moreover, I is feasible if and only if I' is feasible. Furthermore, the instance (T, S, S') is always feasible.*

Using this result, reconfiguration can be performed on a PMR instance $I = (G, S, D)$ to get an equivalent instance $I' = (G, S', D)$ so that S', D have the same underlying vertex set. To do this, find a spanning tree T of G. The $O(n)$ time algorithm guaranteed by Theorem 13 can then compute a desired instance (T, S', D) with S', D having the same set of vertices. Since the moves taking (T, S, S') is feasible, (G, S, S') is feasible; therefore, (G, S, D) is feasible if and only if (G, S', D) is feasible. Given an instance $I = (G, S, D)$ in which S, D have the same underlying set, we call it the *pebble permutation with rotation* problem or PPR. The rest of the section focuses on the feasibility of PPR. Given a PPR instance, we say that two pebbles are *equivalent* if they can exchange locations with no net effect on the locations of other pebbles. A set of pebbles are equivalent if every pair of pebbles from the set are equivalent.

In testing the feasibility of an PPR instance $I = (G, S, D)$, a simple but special case is when G is a cycle. In this case, S and D induce natural cyclic orderings of the pebbles. The following observation is obvious.

Observation 14 *Let $I = (G, S, D)$ be an instance of PPR in which G is a cycle. Then I is feasible if and only if $s_i = d_{(i+k) \bmod p}$ for some fixed natural number k.*

When G is not a cycle, the feasibility test is partitioned into four main cases, depending on the number of pebbles, p, with respect to the number of vertices of G. We assume that G contains at least one TECC since otherwise T is a tree and the problem is a PMT problem.

A. Feasibility test of PPR when $p = n$

When $p = n$, all vertices are occupied by pebbles. Clearly, if a pebble is on a vertex that does not belong to any cycle (i.e., a branch vertex), the pebble cannot move. Therefore, $I = (G, S, D)$ is feasible only if for every branch vertex $v \in V(G)$, $S^{-1}(v) = D^{-1}(v)$. Furthermore, given any TECC C of G, $S^{-1}(C) = D^{-1}(C)$ must also hold, since pebbles cannot move out a TECC. If these conditions hold, the feasibility of I is reduced to feasibilities of $\{(C_i, S|_{S^{-1}(C_i)}, D|_{D^{-1}(C_i)})\}$, in which C_i's are the TECCs of G and $S|_{S^{-1}(C_i)}$ denotes S restricted to the domain $S^{-1}(C_i)$; same applies to $D|_{D^{-1}(C_i)}$. More formally,

Proposition 15 *Let $I = (G, S, D)$ be an instance of PPR with $p = n$. Let $\{C_i\}$ be the set of 2-edge-connected components of G. Then I is feasible if and only if the following holds:*

1) *For all $v \in V(G \backslash (\cup_i C_i))$, $S^{-1}(v) = D^{-1}(v)$.*
2) *For each C_i, $S^{-1}(C) = D^{-1}(C)$.*
3) *For each C_i, the PPR instance $(C_i, S|_{S^{-1}(C_i)}, D|_{D^{-1}(C_i)})$ is feasible.*

Moreover, the feasibility test can be performed in linear time.

PROOF. Finding TECCs of G can be done in $O(|V| + |E|)$ time [13]. Checking whether condition 1 holds takes linear time. For checking condition 2, for each C_i, we first gather $S^{-1}(C_i)$ and for each pebble in $S^{-1}(C_i)$, mark the pebble as belonging to C_i. We can then check whether the pebbles in $D^{-1}(C_i)$ also belong to C_i in linear time. For condition

3, deciding the feasibility of $(C_i, S|_{S^{-1}(C_i)}, D|_{D^{-1}(C_i)})$ can be done using the results from Section III. This check can performed as follows. 1. Check whether C_i is a cycle, which is true if and only if no vertex of C_i has degree more than two. If this is the case, apply Observation 14 to test the feasibility on C_i; 2. Check whether C_i is a cactus with no even cycle. We can verify whether C_i is a cactus as follows: Using depth first search (DFS), detecting cycles of C_i. If C_i is a cactus, then it should assume a "tree" structure shown in Fig. 5; the first cycle that is found must be a leaf cycle. Deleting this cycle (without deleting the vertex that joins this cycle to the rest of C_i) from C_i yields another cactus. Repeating the process tells us whether C_i is a cactus. As we are finding the cycles, we can check whether there is an even cycle. If C_i is indeed a cactus with no even cycle, the possible configurations have two equivalence classes. The subproblem is only infeasible if $S|_{S^{-1}(C_i)}, D|_{D^{-1}(C_i)}$ fall into different equivalence classes, which can be checked by computing the parity of the permutation $\sigma_{S,D}$, restricted to C_i, in linear time; 3. For all other types of C_i, the subproblem is feasible. □

B. Feasibility test of PPR when $p = n - 1$

When $p = n - 1$, nearly all PPR instances, in which G are 2-edge-connected graphs, are feasible.

Lemma 16 *Let $I = (G, S, D)$ be an instance of PPR in which G is 2-edge-connected and not a cycle. If $p < n$, then I is feasible.*

PROOF. By Propositions 7 and 10, $\mathbf{G} \geq \mathbf{A_n}$. That is, there are at most two equivalence classes of configurations, with configurations from different classes differ by a transposition of neighboring pebbles. Since there is at least one empty vertex, viewing that vertex as a "virtual" pebble that can be exchanged with a neighboring pebble in one move, it is then clear that the two configuration classes collapse into a single class. □

Lemma 17 *Let $I = (G, S, D)$ be an instance of PPR in which G, after deleting one (or more) degree 1 vertex (vertices), is a 2-edge-connected graph. If $p < n$, then I is feasible.*

PROOF. Note that by degree 1 vertices, we mean that these vertices have degree 1 in G. Let H be the 2-edge-connected graph after deleting all degree 1 vertices and let $v_1, \ldots, v_k$ be the degree 1 vertices. Let the neighbor of v_i in G be $v_i' \in V(H)$. Since $v \in v_1, \ldots, v_k$ has degree 1, it is attached to H via a single edge. Let H_i be the subgraph of G after deleting all vertices in $v_1, \ldots, v_k$ except v_i. Assume that v_1 is empty initially, we show next that all pebbles occupying H_1 are equivalent. That is, an arbitrary configuration of these pebbles can be achieved.

If H is cycle, the subroutine illustrated in Fig 9 shows how an arbitrary configuration of pebbles can be achieved for a triangle H, which directly generalizes to an arbitrary

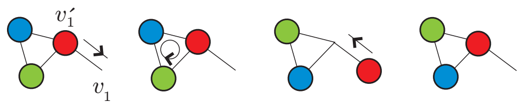

Fig. 9. With one empty vertex, pebbles on a triangle can be arranged to achieve any desired configuration. This generalizes to an arbitrary TECC.

sized cycle. This shows that all pebbles on H_1 fall in the same equivalence class. If H is not a cycle, we can move an arbitrary pebble j from H to v_1. Lemma 16 implies that all pebbles on H are equivalent. Since j is arbitrary, all pebbles on H_1 are equivalent.

Having shown that all pebbles on H_1 are equivalent, we move an arbitrary pebble j to v_1 and empty vertex v_2 (if there is a v_2). Following the same procedure, all pebbles on H_2 are equivalent. Since j is arbitrary, all pebbles on H, v_1, v_2 are equivalent. Inductively, all pebbles on G are equivalent. Therefore, an arbitrary instance I is feasible. □

When there is a single empty vertex on G, it is clear that pebbles can be moved so that the empty vertex is an arbitrary vertex of G. In particular, for any TECC H of G, we can move the pebbles so that a vertex of H is empty. By Lemma 17, all pebbles on H and its distance one neighboring vertices fall in the same equivalence class. We now show that the feasibility of the case of $p = n - 1$ can be decided in linear time.

Proposition 18 *Let $I = (G, S, D)$ be an instance of PPR in which $p = n - 1$ and G is not a cycle. The feasibility of I can be decided in linear time.*

PROOF. We start with pebble configuration S and group the pebbles into equivalence classes. Without loss of generality, assume that S leaves a vertex of a TECC, say H, unoccupied. By Lemma 17, all pebbles on H and its distance 1 neighbors belong to the same equivalence class, say $h_{S,1}$. Now, check whether any pebble in $h_{S,1}$ is on some other TECC $H' \neq H$. If that is the case, all pebbles on H' and its distance 1 neighbors are also equivalent and belong to $h_{S,1}$. When no more pebbles can be added to $h_{S,1}$ this way, $h_{S,1}$ is completely defined.

Let v be a vertex neighboring a vertex occupied by a pebble from $h_{S,1}$ (v itself is not occupied by a pebble in $h_{S,1}$), if v is not a TECC vertex, the pebble currently on v cannot be move to a TECC and therefore is not equivalent to any other pebble. The pebble then gets its own equivalence class, say $h_{S,2}$. If v belongs to a TECC, say H_v, then all pebbles on H_v and all H_v's distance 1 neighbors that are not yet classified belong to $h_{S,2}$; $h_{S,2}$ is then expanded similarly to $h_{S,1}$. At this point, the procedures given so far apply to partition all pebbles into equivalence classes. It is not hard to see the algorithm takes linear time to complete using breadth first or depth first search, treating each TECC as a whole.

As the start configuration S is being classified, the same is done to D. In particular, if a set of pebbles of S belongs to an equivalence class $h_{S,i}$, then the pebbles of D occupying the same set of vertices get assigned to the class $h_{D,i}$. The instance I is feasible if and only if $h_{S,i} = h_{D,i}$ for all i (this can be done in linear time as we have shown in checking the second condition in Proposition 15). □

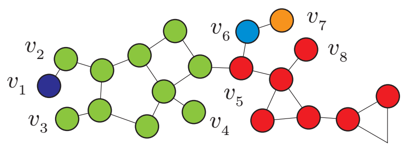

Fig. 10. An example of the case $p = n-1$. The pebbles are put into 5 different equivalence classes, distinguished by different colors.

Fig. 10 provides an example of applying the above procedure to a given pebble configuration, which partitions the pebbles into 5 equivalence classes.

C. *Feasibility test of PPR when $p < n(TECCs)$*

We denote by $n(TECCs)$ the number of vertices of all TECCs of G. When $p < n(TECCs)$, the instance I is almost always feasible.

Theorem 19 *Let $I = (G,S,D)$ be an instance of PPR in which G is not a cycle. If $p < n(TECCs)$, then I is feasible.*

PROOF. Since the number of pebbles are not enough to occupy all TECC vertices, we can update configuration S to a new one S' such that all pebbles are on TECC vertices. Repeating the same moves over the configuration D to get D' (i.e., if we move a pebble from v_i to v_j in the initial pebble configuration, we move the corresponding pebble from v_i to v_j in the final pebble configuration). After this process is complete, the updated start and final configurations again occupy the same set of vertices; (G,S,D) is feasible if and only if the (G,S',D') is feasible. In the rest of the proof we show that (G,S',D') is feasible.

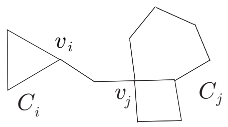

Fig. 11. A graph with two TECCs.

Since not all TECC vertices are occupied in S', at least one TECC, say C_i, has an empty vertex. By Lamma 17, all pebbles on C_i are equivalent. Now let C_j be another TECC joined to C_i via a single branch (see Fig. 11 for an example). Since any pebble on C_j can be moved to vertex v_j via a proper sequence of rotations, it is then possible to exchange any pair of pebbles p_1 on C_i and p_2 on C_j: Move p_2 to v_j, empty v_i, move p_2 to v_i, rotate p_1 to v_i, and move it to v_j. Via induction, any pair of pebbles on G can be exchanged, without affecting the current configuration of other pebbles. Given this procedure, we can iteratively arrange each pebble i, starting from pebble 1, by exchanging pebble i with some other pebble occupying i's vertex in D'. With up to $p-1$ exchanges, all pebbles can be arranged to their desired final configurations. □

D. *Feasibility test of PPR when $n(TECCs) \le p < n-1$*

For this last case, given an PPR instance, (G,S,D), we first move pebbles in S,D so that vertices of all TECCs are occupied. To perform this in linear time, a "fake" goal configuration D_f is created with p pebbles such that all TECCs are full occupied, in an arbitrary order (this is doable because $n(TECCs) \le p < n-1$). Using a spanning tree T of G and apply Theorem 13 to $(T,S,D_f),(T,D,D_f)$, we get two new instances (T,S',D_f), (T,D',D_f) with the property that S',D',D_f all occupy the same set of vertices and (T,S,S'), (T,D,D') are both feasible. Thus, we obtain a new PPR instance (G,S',D'), which is feasible if and only if (G,S,D) is, with the additional property that vertices of all TECCs are occupied. For convenience, we call an instance (G,S,D) of PPR in which $n(TECCs) \le p < n-1$ and all TECC vertices are occupied a *rearranged pebble permutation* problem, or RPP.

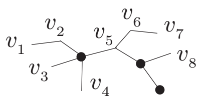

Fig. 12. The skeleton tree after contracting the graph from Fig. 10; the black dots are the composite vertices.

Next, we contract G to get a *skeleton tree*, T_G, by collapsing each TECC into a *composite vertex*; other vertices and edges are left intact. For example, the graph from Fig. 10 have the skeleton tree shown in Fig. 12. This procedure induces a natural map f_T that takes any subgraph H of G to $f_T(H)$ as a subgraph of T_G (via mapping all vertices belonging to the same TECC of G to a composite vertex of T_G and non-composite vertices of G to non-composite vertices of T). Given an instance (G,S,D) of RPP with $p < n-1$ pebbles, all pebbles on the same TECC are equivalent by Lemma 17. This induces a problem instance (T_G,S',D') in which all pebbles (in S and D) on the same TECC of G are combined into a *composite* pebble (in S' and D'). Given two vertices u,v in a graph, $u \rightsquigarrow v$ denotes a (shortest) path between u,v. Such a path is unique when the graph is a tree. By all vertices on (resp. in) $u \rightsquigarrow v$, we mean vertices of $u \rightsquigarrow v$ including (resp. excluding) u,v. We now extend Lemma 6 from [1] to RPP.

Lemma 20 *Let (G,S,D) be an instance of RPP in which G is not a cycle and $n(TECCs) \le p < n-1$. Let u,v,w be vertices of G such that the path between u,v and the path between v,w are not edge disjoint. Assume u,v are occupied by pebbles and moves exist that take S to a new configuration in which pebble $S^{-1}(u)$ is moved to v and $S^{-1}(v)$ is moved to w. Then S can be taken to an configuration S' in which S,S' are the same except pebbles on u,v are exchanged.*

PROOF. For convenience, let $p_1 := S^{-1}(u)$ and $p_2 := S^{-1}(v)$. Let the overlapping part of $u \rightsquigarrow v$ and $v \rightsquigarrow w$ be $y \rightsquigarrow v$. Let the sequence of moves that take p_1 to v and p_2 to w be represented as $X = \langle S = S_0, S_1, \ldots, D\rangle$. If it is possible to move p_1, p_2 to the same TECC, then clearly the locations of p_1, p_2 can be exchanged on the TECC without changing any other pebble's configuration. Reversing earlier moves then exchanges p_1, p_2 on u, v. For the rest of this proof, we assume that p_1, p_2 can never occupy vertices from the same TECC. Note that this implies hat p_1, p_2 can never occupy vertices of the same TECC in different configurations originated from S; in particular, no vertex on $y \rightsquigarrow v$ can be on a TECC. To see this, if p_1, p_2 both reach a TECC H in some (possibly different) configurations in X, assume without loss of generality that p_1 reaches H first. Since all pebbles on H are equivalent and H contains at least three vertices, p_1 can always stay on H: Suppose X at some point wants to move p_1 outside of H. If p_1 is the only pebble on H, p_1 does not hinder any other pebbles from moving through H and moving p_1 out will only crowd the rest of G, making further pebble movements outside H harder. If p_1 is not the only pebble on H, we may pick any pebble on H to leave H instead of p_1. Then p_2 will eventually reach H with p_1 still on H, allowing them to exchange.

For the case in which p_1, p_2 never visits the same TECC of G, let W denote the graph formed by the vertices and edges traveled by p_1, p_2 as they move along the sequence of configurations in X. Let $T_W = f_T(W)$. If T_W contains composite vertices that are not leaves of T_W, let z be such a composite vertex and H_z be the TECC corresponding to z in G. Let $G(H_z, v)$ denote the connected component of G containing v after deleting H_z and let $\overline{G}(H_z, v)$ denote rest of the components. By assumption, only one of p_1 or p_2 may visit H_z. Assume it is p_1 (the case of p_2 is similar), then p_2 can only visit vertices of $G(H_z, v)$; in fact the entire path $v \rightsquigarrow w$ is within $G(H_z, v)$. Using the same argument from the previous paragraph, X can be modified so that p_1 does not visit vertices of $\overline{G}(H_z, v)$, unless $u \in \overline{G}(H_z, v)$. In this case, however, p_1 is equivalent to any pebble that is initially on H_z; the lemma holds if an only if a pebble initially on H_z in S can move to v and p_2 can move to w. Via induction, it must be possible for some pebbles p_1', equivalent to p_1, and p_2 to move from some u' to v and v to some w', respectively, where $y \rightsquigarrow v$ is contained within $u' \rightsquigarrow v$ and $v \rightsquigarrow w'$. Further more, p_1', p_2 do not "pass through" any TECC of G.

We may then assume that from the beginning, T_W has only composite vertices that are leaves. Denote the branch of G containing y as T_y. Since p_1, p_2 may still visit some TECCs, let T_y' denote the tree containing T_y as well as the vertices of these TECCs (visited by p_1 or p_2) that are (distance 1) neighbors of T_y. Since the labels of pebbles other than p_1, p_2 have no effect on moving p_1, p_2, we may assume pebbles other than p_1, p_2 are unlabeled (indistinguishable). It can be shown that unlabeled pebbles outside of T_y' never need to move to T_y': If an unlabeled pebble moves from outside T_y' and stays on T_y' it only makes moving p_1, p_2 less feasible; if an unlabeled pebble moves from one vertex outside T_y' to another vertex outside T_y' via T_y, it does not help the feasibility of moving p_1, p_2 on T_y'. Thus, unlabeled pebbles may only move away from T_y' and they should never come back. Therefore, we may first take the unlabeled pebbles that will leave T_y' and move them outside T_y' in the beginning. After these steps, the initial problem is reduced to moving p_1 from u to v and p_2 from v to w on the tree T_y'; by Lemma 6 from [1], p_1, p_2 are equivalent. Note that this implies that if p_1 (resp. p_2) can visit a TECC, then p_2 (resp. p_1) can visit that TECC as well; it is not possible that a given TECC can only be visited by one of the pebbles from p_1, p_2. □

Lemma 20 leads to a generalized version of Theorem 4 from [1] to RPP, given below. We omit the proof since it is nearly identical (we need extended versions of Corollary 1 and 2 from [1], which can be easily proved in the same way Lemma 20 is proved).

Theorem 21 *An RPP instance, (G, S, D), in which G is not a cycle and $n(TECCs) \leq p < n-1$, is feasible if and only if the individual exchanges between pebble i and $S^{-1}(D(i))$, $1 \leq i \leq p$, can be performed using moves without affecting the configurations of any other pebble.*

By Theorem 21, if an instance of RPP, $I = (G, S, D)$, is feasible, then pebbles i and $\sigma_{S,D}(i) = S^{-1}(D(i))$ can be exchanged with no net effect on other pebbles. This enables a feasibility test of RPP problems (and therefore, PMR problems): Vertices occupied by pebbles are partitioned into equivalence classes such that two pebbles can be exchanged if and only if the vertices occupied by them belong to the same equivalence class. In fact, we apply the *Mark* algorithm from [1] on the skeleton tree T_G without any change at the pseudocode level (see [1] for the simple algorithm description); the main level level difference is how to check whether two adjacent pebbles are equivalent (Lemma 8 from [1]).

Before stating our version of the lemma, some notations are in order. We assume that we work with an arbitrary RPP instance $I = (G, S, D)$ in which G is not a cycle and $n(TECCs) \leq p < n-1$. Let $I' = (T_G, S', D')$ be the induced instance described earlier in which T_G is G's skeleton tree. A *fork* vertex of T_G is a vertex of degree at least 3 that is not a composite vertex. $F(u)$ is the set of connected components of T_G after deleting the vertex u. $T(u, v)$ is the tree of $F(u)$ containing the vertex v; $\overline{T}(u, v)$ is the rest of $F(u)$. For two vertices $u, v \in V(T_G)$, $d(u, v)$ is the length of $u \rightsquigarrow v$. In the lemmas that follow, only start configuration S' is operated on; same procedure can be applied to D. First we need a version of Corollary 3 from [1] to account for composite vertices; we omit the essentially same proof but point out that although both fork and composite vertices can help two pebbles switch locations, composite vertex can do so with one fewer empty vertex.

Lemma 22 *Let $p_1 := S'^{-1}(u)$, $p_2 := S'^{-1}(v)$ for $u, v \in V(T_G)$ such that $u \rightsquigarrow v$ contains no other pebbles; all vertices on*

$u \rightsquigarrow v$ are of degree 2. Let w be a composite or fork vertex to u such that u is in $w \rightsquigarrow v$. If w is a composite (resp. fork) vertex, then $T(u,w)$ has no more than $d(w,u)$ (resp. $d(w,u)+1$) empty vertices. Let w' be the closest composite or fork vertex to v such that v is in $w' \rightsquigarrow u$ satisfying similar properties as w. Then u,v are not equivalent.

Lemma 23 *Let $p_1 := S'^{-1}(u)$, $p_2 := S'^{-1}(v)$ for some $u,v \in V(T_G)$ such that $u \rightsquigarrow v$ contains no other pebbles. Then p_1, p_2 are equivalent with respect to x if and only if at least one of the following conditions holds:*

1. There exists fork vertex w in $u \rightsquigarrow v$ such that both $T(w,u), T(w,v)$ are not full or at least one other tree of $F(w)$ is not full.
2. Let w be a composite vertex such that u is in $w \rightsquigarrow v$ and no other fork vertex or composite vertex is in $w \rightsquigarrow u$. There exists such a w that $T(u,w)$ has $d(w,u)+1$ empty vertices.
3. Similar to 2 but with u,v switched.
4. Let w be a fork vertex such that u is in $w \rightsquigarrow v$ and no other fork vertex or composite vertex is in $w \rightsquigarrow u$. There exists such a w that $T(u,w)$ has $d(w,u)+2$ empty vertices.
5. Similar to 4 but with u,v switched.
6. Vertex u is a fork vertex. Then at least two trees of $F(u)$ has empty vertices or there are at least two empty vertices outside $T(u,v)$.
7. Similar to 6 but with u,v switched.
8. Vertex u is a composite vertex. Then at least one tree of $\overline{T}(u,v)$ has an empty vertex.
9. Similar to 8 but with u,v switched.

PROOF SKETCH. The proof is adapted from that of Lemma 8 from [1] so some repetitive details are omitted; for these see [1]. Since the sufficiency of the conditions can be easily checked by constructing plans that exchange p_1, p_2, only necessity is shown, via contradiction. Assume that u,v are exchangeable without configuration S satisfying any of the conditions 1-9. First consider the case in which there is no fork vertex in $u \rightsquigarrow v$ and u,v are not fork or composite vertices; these assumptions forbids conditions 1 and 6-9. If conditions 2-5 do not hold, the condition from Lemma 22 is true, thus u,v cannot be equivalent.

For the case in which no fork vertex in $u \rightsquigarrow v$ but u or v (possibly both) is a fork or composite vertex, the proof from Lemma 8 from [1] applies with little change to show that u,v are not equivalent unless one of conditions 2-9 holds: If conditions 2-5 do not hold, this means that p_1, p_2 must use u or v as a "hub" for switching locations; traveling beyond distance 1 from $u \rightsquigarrow v$ will not help u,v to switch. On the other hand, if conditions 6-9 do not hold, u or v cannot serve as the hub that enables u,v to switch. Furthermore, if conditions 6-9 do not hold, reconfiguration of pebbles will not make conditions 2-5, previously invalid, become valid.

This leaves the case in which conditions 2-9 do not hold, which means that u,v cannot switch on $\overline{T}(u,v)$ nor $\overline{T}(v,u)$. Since there is no pebble in $u \rightsquigarrow v$, the vertices in $u \rightsquigarrow v$ cannot be composite vertices. The same proof from Lemma 8 from [1] then shows that unless condition 1 is met, u,v cannot be equivalent. □

With Lemma 23, all criteria needed for the *Mark* algorithm from [1], in particular Observations 1-4, continue to hold on T_G without change. Since *Mark* is not changed, its running time is linear if deciding whether two adjacent pebbles are equivalent can be performed in (amortized) constant time. For this to hold, for an arbitrary tree $T(u,w)$, we need to know whether $T(u,w)$ has 0, 1, 2 holes and whether the fork or composite vertex of $T(u,w)$ closest to u allows u and another vertex v in $T(u,w)$ to exchange (i.e., $T(u,w)$ should have enough empty vertices). These data can be precomputed in $O(|V|+|E|)$ time using two depth firth traversals over the tree T_G. At this point, it is not hard to see that this linear decision algorithm easily turns into an algorithm that computes a feasible solution to a PPR instance. Our complexity analysis shows that a feasible solution can be computed in $O(|E|)$ if a high level plan is required (computes a corresponding RPP instance, checks feasibility, and outputs the permutation pairs for exchanges) and $O(n^3)$ if step by step output is required (each exchange can be done in $O(n^2)$ moves produced by a fixed formula).

E. Linear time feasibility test for PMG

Using the same breakdown based on the number of pebbles, we can also test feasibility of the pebble motion problem for general graph in which the rotation move is not allowed (the PMG problem). Here we provide a brief description of this algorithm, highlighting its difference from the feasibility algorithm for PMR problem at a high level.

Given a PMG instance, using the procedure described in the first paragraph of Subsection IV-D, we can obtain in linear time an instance $I = (G,S,D)$ that is feasible if and only if the initial instance is feasible. Moreover, S,D now contain the same set of vertices of G and we can choose D_f (see first paragraph of Subsection IV-D) so that TECC vertices are occupied first. This ensures that: 1. When $p \leq n(TECCs)$, S,D contain only TECC vertices; 2. When $p \geq n(TECCs)$, S,D contain all TECC vertices. Assume for the rest of this subsection that we work with such an instance.

The case of $p = n$ is trivial. Same is true for the case in which G is a cycle or tree; assume these conditions do not hold. When $p = n-1$, given a TECC H of G that is not a cycle, one of its vertex can be emptied, leaving $n(H)-1$ pebbles on H. By [7], there $n(H)-1$ pebbles are either equivalent or fall into two equivalence classes (except when H is the T_0 graph [7], but this only introduces a constant number of additional operations). When H is a cycle, each of the $n(H)-1$ pebbles has its own equivalence class. These facts suggest that no two pebbles on two different TECCs are equivalent and no two pebbles on different non-TECC vertices are equivalent. The case of $p = n-1$ then becomes deciding pebble equivalence on individual TECCs.

When there are two or more empty vertices, after moving out one pebble, say p_1, from a TECC H of G, the rest $n(H)-1$ pebbles are all equivalent: Since there is another vacant vertex, we can move another pebble, say p_2, out of

H. The rest $n(H)-2$ pebbles are then equivalent. It is clear that we can move p_2 back and move at least another pebble different from p_2 out of H; therefore, all $n(H)-1$ pebbles are equivalent. This observation partitions all pebbles on H into (at most) two equivalence classes: Pebble p_1 and the rest $n(H)-1$ pebbles. If $p \le n(TECCs)-3$, then the instance is feasible following the argument used in showing the case of $p < n(TECCs)$ for the PMR problem. Alternatively, if $p = n(TECCs)-2$ and there is a single TECC, the instance is also feasible.

The last main case is when $n(TECCs)-2 \le p < n-1$. For this case, a version of Lemma 20 can be stated and proved using the same proof technique. In particular, if two pebbles p_1, p_2 can move from u to v and from v to w, respectively, with $u \rightsquigarrow v$ and $v \rightsquigarrow w$ sharing at least an edge, then we can show that a pebble p_1' equivalent to p_1 exists so that p_1', p_2 can move from u' to v and from v to w', respectively, with $u' \rightsquigarrow v$ and $v \rightsquigarrow w'$ sharing at least an edge. Furthermore, p_1', p_2 need not travel beyond any TECC. A similar version of Lemma 23 can then be obtained (we need a few additional conditions, including a condition to decide whether the two classes of pebbles on the same TECC are equivalent), and the feasibility of the instance can be decided using the *Mark* algorithm based on this. We conclude this section with the following theorem.

Theorem 24 *The feasibility of PMR/PMG problems can be decided in linear time.*

V. Other Aspects

A. *Computational Complexity of* **TOSPM**

Proposition 25 (NP-hardness) **TOSPM** *is NP-hard.*

Proof. We reduce the optimal multi-agent path planning problem formulation in [12] (denote this problem *parallel pebble motion*, or **PPM** for short) to **TOSPM**. A decision version of **PPM** can be stated simply as Σ/η: Σ is an instance of **PPM** and η is a natural number. The NP-hardness proof in [12] reduces an instance of the SAT to an instance of Σ/η and shows that Σ admits a solution with completion time (denoted as *makespan* in [12]) no more than η if and only the SAT problem is satisfiable. We make the observation that the constructed Σ does not admit a solution with completion time of η or less if the agents were to rotate along a fully occupied cycle. That is, Σ/η, constructed from a SAT instance, is also an instance of **TOSPM** with $k=\eta$, in which a (yes) certificate cannot contain rotations along fully occupied cycles. Therefore, **TOSPM** is also NP-hard. □

Since **TOSPM** is in NP because PPR is in NP, it follows that **TOSPM** is NP-complete.

B. *Some Tighter Diameter Bounds*

In this subsection, we make finer characterizations on group diameters of two subclasses of 2-edge-connected graphs: Graphs with bounded number of cycles and grids. Note that this assumes all vertices are occupied by pebbles

Graphs with Bounded Number of Cycles. By *bounded* number of cycles, we mean that the a graph G cannot have more cycles than a predetermined number. From Proposition 7 and 10, if G is 2-edge-connected and not a single cycle, $diam(\mathbf{G}) \le O(n^2)$. Here we provide a lower bound on $diam(\mathbf{G})$.

Proposition 26 (General Lower Bound) *If a graph G is 2-edge-connected, is not a single cycle, and has no more than $c < \infty$ cycles, $diam(\mathbf{G}) = \Omega(n \log n)$.*

Proof. $\mathbf{G} \ge \mathbf{A_n}$ by Proposition 7 and 10. With Stirling's approximation, we have

$$|\mathbf{G}| \ge \frac{n!}{2} \ge \sqrt{\frac{\pi n}{2}}(\frac{n}{e})^n.$$

Since G has no more than c cycles, $|\mathscr{C}| \le c!$ and $\mathscr{G}| \le 2^c|\mathscr{C}| =: M$. To see that the later is true, note that each set of disjoint cycles $C \in \mathscr{C}$ cannot produce more than 2^c distinct generators. By Lemma 4, using generator products of length k, at most M^{k+1} elements of $\mathbf{G}$ are reachable. To reach all elements of $\mathbf{G}$ from the identity, a necessary condition is

$$M^{k+1} \ge |\mathbf{G}| \ge (\frac{n}{e})^n,$$

which implies that to reach certain element of $\mathbf{G}$, a generator product of length at least $\log_M |\mathbf{G}| = \Omega(n \log n)$ is required. □

It is clear that for an arbitrary fixed $c < \infty$, graphs with up to c unique cycles form an infinite set. The $\Omega(n \log n)$ lower bound suggests that the $O(n^2)$ upper bound is reasonably tight. An interesting open question is whether the $n \log n$ lower bound is tight for 2-edge-connected graphs with bounded number of generators since it appears possible to beat the $O(n^2)$ upper bound using clever compositions of cycle generators.

Grid graphs. Our attention on grid graphs focuses on two dimensional $m(row) \times n(column)$ grids. For convenience, for pebbles numbered from 1 to mn, we assume that the goal configuration is always the shuffled row-major ordering of the pebbles (i.e., if we represent the rows as n-tuples ordered from left to right, then the k-th row from the top reads $((k-1)n+1, (k-1)n+2, \ldots, kn)$). Given such a grid graph G, a lower bound on $diam(\mathbf{G})$ is easy to obtain.

Lemma 27 (Grid, Lower Bound) *For a $m \times n$ grid graph G, $diam(\mathbf{G}) = \Omega(m+n)$.*

Proof. Let the initial configuration have pebble 1 placed at the lower right corner of the grid, then it takes at least $m+n-2$ moves to get it to the top left corner. □

Somewhat surprisingly, for grids, the upper bound on the group diameter matches the lower bound asymptotically, due to the presence of small cycles (4-cycles in this case).

Proposition 28 (Grid, Upper Bound) *For a $m \times n$ grid graph G, $diam(\mathbf{G}) = \Theta(m+n)$.*

PROOF. Given the previous lemma, we are left to show that $diam(\mathbf{G}) = O(m+n)$. For this task, we assume that $n \geq m$ and work with a $n \times n$ grid instead.

Our first claim is that parallel bubble sort (i.e., any two adjacent pebbles on G may exchange locations in one step provided that these two pebbles are not involved in other exchanges simultaneously) on an $n \times n$ grid can be done in $O(n)$ parallel steps, which is achievable in various ways. For example, Theorem 8.1 in [14] shows that Batcher's bitonic merge algorithm [3] can perform such sorting tasks in about $15n$ parallel operations. Note that the sorting method does better than $15n$ when applied to a $m \times n$ grid with $m \leq n$.

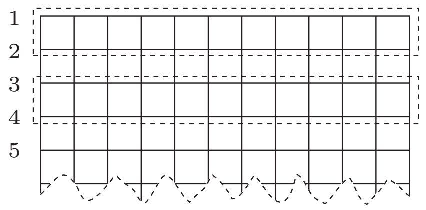

Fig. 13. Partition a grid into 2-row strips.

Our next claim is that each parallel sorting (bubbling) step can be completed using constant number of moves (this step is insensitive to whether the grid is a square or not). In the bitonic merge algorithm, each parallel move is either all exchanges vertically or all exchanges horizontally. Without loss of generality, assume that we need to exchange some pairs of pebbles horizontally. To complete this task, we pair the rows to form 2-row strips and operates within these strips (see Fig. 13). Since pebbles only exchange horizontally, no vertex exchange happens across strips. If we can show that exchanging pebbles within each strip can be completed using constant number of moves, we are done (note that when n is odd, an extra strip from row $n-1$ and row n must be sorted afterwards).

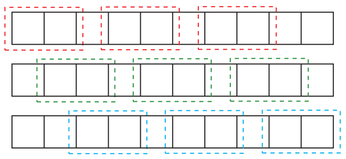

Fig. 14. Partition of a 2×11 strip into 2×3 blocks: Three partitions are enough to ensure any two horizontally adjacent vertices will appear in at least one partitioned 2×3 block, allowing them to exchange.

For each strip, we partition it into consecutive but disjoint 2×3 blocks, first starting from the leftmost column. Doing this at most three times, each time shifting the starting block by one vertex to the right, ensures that all pebbles adjacent to each other horizontally will appear in a same 2×3 block at least once, allowing them to be exchanged (see Fig. 14). Since exchanging any two vertices horizontally is achievable within a 2×3 block using 3 moves

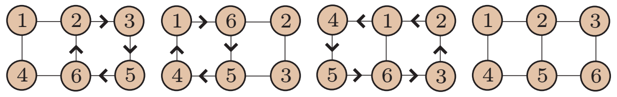

Fig. 15. A 3-step procedure for exchanging vertices 5 and 6 within a 2×3 block.

according to the procedure illustrated in Fig. 15, each strip can be ordered as desired in a constant number of moves. □

The proof of Proposition 28 readily extends to other types of graphs such as k-dimensional grids for fixed k or hypercubes provided that a parallel sorting scheme can be implemented using local exchanges. For example, an $O(n^2)$ parallel sorting algorithm is given in [9] for hypercube 2^n, yielding a $O(n^2)$ diameter for the group generated by rotations on the hypercube with 2^n vertices. For $n \times n$ grids, Proposition 28 guarantees the existence of a path set that is within a constant factor of a time optimal path set. Such a near optimal path set can also be computed efficiently. For each of the $O(n)$ parallel exchange operations, up to $n^2/2$ exchanges are required, yielding an overall time complexity of $O(n^3)$, since an exchange can be performed locally in constant time. Alternatively, the computation can be completed in $O(n)$ steps using n^2 processors with proper interconnections.

REFERENCES

[1] V. Auletta, A. Monti, M. Parente, and P. Persiano. A linear-time algorithm for the feasbility of pebble motion on trees. *Algorithmica*, 23:223–245, 1999.

[2] L. Babai, R. Beals, and Á. Seress. On the Diameter of the Symmetric Group: Polynomial Bounds. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '04)*, pages 1108–1112, 2004.

[3] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference (AFIPS '68 (Spring))*, pages 307–314, 1968.

[4] J. R. Driscoll and M. L. Furst. On the diameter of permutation groups. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC '83)*, pages 152–160, 1983.

[5] J. R. Driscoll and M. L. Furst. Computing short generator sequences. *Information and Computation*, 72(2):117–132, February 1987.

[6] O. Goldreich. Finding the shortest move-sequence in the graph-generalized 15-puzzle is np-hard. 1984. Laboratory for Computer Science, Massachusetts Institute of Technology, unpublished manuscript.

[7] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS '84)*, pages 241–250, 1984.

[8] S. Loyd. *Mathematical Puzzles of Sam Loyd*. Dover, New York, 1959.

[9] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1168–1176, 1995.

[10] D. Ratner and M. Warmuth. The (n^2-1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10:111–137, 1990.

[11] E. W. Story. Note on the '15' puzzle. *American Journal of Mathematics*, 2:399–404, 1879.

[12] P. Surynek. An optimization variant of multi-robot path planning is intractable. In *The Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1261–1263, 2010.

[13] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):140–160, 1972.

[14] C. D. Thompson and H. T. Kung. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 20(4):263–271, 1977.

[15] R. M. Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory (B)*, 16:86–96, 1974.